# On the (Im)possibility of Aggregate Message Authentication Codes

Aldar C-F. Chan
National University of Singapore

Claude Castelluccia
INRIA

***Abstract*—In data aggregation, multiple source nodes send their data to a sink along a concast tree with aggregation done en route so that the sink can obtain the aggregate (which could be the sum, average, etc.) of all these data. End-to-end privacy and aggregate integrity are the two main goals of secure data aggregation. While the privacy goal has been widely studied, providing end-to-end aggregate integrity in the presence of possibly compromised aggregating nodes remains largely an open problem. Message Authentication Codes (MAC) are commonly used to provide end-to-end data integrity in two party settings. Natural extensions of MAC for the data aggregation scenario are considered. It is shown that a straightforward and intuitive refinement of the MAC security model (for the data aggregation setting) is not achievable. A weaker security notion is proposed; whether this notion is achievable remains unclear.**

## I. Introduction

Due to the constraint on power consumption (for wireless transmission), data aggregation emerges as the de facto paradigm for performing aggregate queries over a wireless sensor network. In this actively studied paradigm, a concast tree rooted at a sink to connect all the reporting sensor nodes is usually formed for each query and aggregation is performed en route, that is, each sensor node aggregates all results from its child nodes in the concast tree and only passes the resulting aggregate to its parent upstream. End-to-end privacy and aggregate integrity/authenticity are the two main security goals for such a data aggregation paradigm. In brief, privacy (regardless of the information leakage due to the correlation among sensor measurements) ensures that nobody other than the sink could learn considerable information about the final aggregate even if he might control any subset of sensor nodes while aggregate authentication assures that any manipulation of the final aggregate by an adversary beyond what is achievable through direct injection of data at compromised nodes under his control will be detected at the sink.

End-to-end privacy of data aggregation has been widely studied in a variant called concealed data aggregation (CDA) [1], [4], [7], [12]. Chan and Castelluccia [5] extend the security model for semantic security, a standard notion of privacy for encryption schemes (both symmetric and asymmetric), to cover the CDA setting. It is natural to ask whether the common notions of integrity or authenticity in cryptography can be extended for the data aggregation scenario in a similar fashion as in CDA. There are a number of constructions to protect aggregate integrity in data aggregation [6], [9], [11], [13] but all these schemes cannot provide end-to-end integrity or only a single layer of aggregation is considered. All proposed schemes for aggregate integrity protection in hierarchical data aggregation require a call-back (by the sink or intermediate aggregating nodes or both) to the downstream sensor nodes (which have performed aggregation) in order to verify the integrity of the aggregate. It is fair to say providing end-to-end aggregate integrity in data aggregation with a multi-level tree remains an open problem. This paper investigates whether such a goal of end-to-end aggregate integrity is achievable.

End-to-end integrity in one-to-one communication is usually provided through the use of Message Authentication Codes (MAC) such as [2], [10]. That is, the two users in communication share a common secret key; each message sent is attached with a tag which is computed using the secret key and the message as input; the integrity of a received message is verified by re-computing the tag based on the received message and the secret key. It is natural to ask whether such a MAC primitive can be extended to cover the data aggregation scenario so as to provide end-to-end aggregate integrity/authentication. This paper studies possible extensions of the MAC security model, collectively called Aggregate Message Authentication Code (AMAC) in this paper.

This paper shows that a straightforward extension of MAC for the aggregation scenario could not be achievable, that is, no secure primitive could be constructed to achieve this AMAC security notion. Then a weaker security notion is proposed. This weakened notion guarantees that what a compromised node can best achieve in deviating the final aggregate is through injecting data using the captured secret keys at compromised nodes. Nevertheless, whether this weakened security notion is achievable remains unclear. This paper shows that if an AMAC primitive achieving this notion exists, it can be used to construct a kind of IND-CCA2-secure CDA; however, no construction of the latter has been proposed in the literature so far. If this type of IND-CCA2-secure CDA does not exist, then the weakened AMAC notion is not achievable either.

### A. Aggregate Authentication vs. Aggregatable Authentication

It should be noted that the problem of aggregate authentication considered in this paper is different from the problem considered in aggregate signatures [3]; more precisely, the latter should be called aggregatable signatures instead. In aggregate authentication (considered in this paper), it is the messages themselves being aggregated and hence the original messages are not available for verification, whereas, in aggregate signatures, the signatures for different messages are aggregated and all the signed messages have to be distinct and available to the verification algorithm in order to verify the validity of an aggregate signature.

In fact, constructing aggregatable MAC which supports tag aggregation is trivial. We can simply take exclusive-OR on all

the MAC tags as in [6]. The idea is as follows: suppose there are $l$ senders each sharing a secret key $k_i$ with a receiver and having a message $m_i$ to be sent to the receiver; to ensure the integrity of their messages, each sender can generate a MAC tag $t_i$ on $m_i$ using the secret key $k_i$ and send out $(m_i, t_i)$; let $\oplus$ denote bitwise exclusive OR; at the receiver, the aggregated tag $T = t_1 \oplus t_2 \oplus ... \oplus t_l$ is sufficient (that is, as good as having all the $t_i$'s) to guarantee the integrity of each message in the series: $m_1, m_2, ..., m_l$. Note that along with $T$ and $k_i$'s, the message $m_1, m_2, ..., m_l$ are still needed at the receiver for verification. It can be shown that this construction achieves existential unforgeability against chosen message attack.

### B. Our Contributions

The main contribution of this paper is two-fold: First, we give a security model for end-to-end aggregate integrity (which may be achievable) in secure data aggregation. It is the first formal treatment to end-to-end integrity in secure data aggregation. Second, we relate this notion for end-to-end integrity in data aggregation with that for end-to-end privacy.

## II. Definitions

A separation of the privacy and the aggregate integrity goals is adopted through two primitives, namely, CDA and AMAC.

**Notations.** We denote by $z \leftarrow A(x, y, \ldots)$ the experiment of running a probabilistic algorithm $A$ on inputs $x, y \ldots$, generating output $z$. We denote by $\{A(x, y, \ldots)\}$ the probability distribution induced by the output of $A$. As usual, PPT denote probabilistic poly-time. An empty set is always denoted by $\phi$.

### A. CDA Syntax

A typical CDA scheme includes a sink $R$ and a set $U$ of $n$ source nodes (which are usually sensor nodes) where $U = \{s_i : 1 \leq i \leq n\}$. Denote the set of source identities by $ID$; in the simplest case, $ID = [1, n]$. In the following discussion, $hdr \subseteq ID$ is a header indicating the source nodes contributing to an encrypted aggregate. Given a security parameter $\lambda$, a CDA scheme consists of the following polynomial time algorithms.

- Key Generation ($\mathsf{KG}$). Let $\mathsf{KG}(1^\lambda, n) \rightarrow (dk, ek_1, ek_2, \ldots, ek_n)$ be a probabilistic algorithm. Then, $ek_i$ (with $1 \leq i \leq n$) is the encryption key assigned to source node $s_i$ and $dk$ is the corresponding decryption key given to the sink $R$.
- Encryption ($\mathsf{E}$). $\mathsf{E}_{ek_i}(m_i) \rightarrow (hdr_i, c_i)$ is a probabilistic encryption algorithm taking a plaintext $m_i$ and an encryption key $ek_i$ as input to generate a ciphertext $c_i$ and a header $hdr_i \subset ID$. Here $hdr_i$ indicates the identity of the source node performing the encryption; if the identity is $i$, then $hdr_i = \{i\}$. *We sometimes denote the encryption function by $\mathsf{E}_{ek_i}(m_i; r)$ to explicitly show by a string $r$ the random coins used in the encryption process.*
- Decryption ($\mathsf{D}$). Given an encrypted aggregate $c$ and its header $hdr \subseteq ID$ (which indicates the source nodes included in the aggregation), $\mathsf{D}_{dk}(hdr, c) \rightarrow m/\perp$ is a deterministic algorithm which takes the decryption key $dk$, $hdr$ and $c$ as input and returns the plaintext aggregate $m$ or possibly $\perp$ if $c$ is an invalid ciphertext.
- Aggregation ($\mathsf{Agg}$). With a specified aggregation function $f$, the aggregation algorithm $\mathsf{Agg}_f(hdr_i, hdr_j, c_i, c_j) \rightarrow (hdr_l, c_l)$ aggregates two encrypted aggregates $c_i$ and $c_j$ with headers $hdr_i$ and $hdr_j$ respectively (where $hdr_i \cap hdr_j = \phi$) to create a combined aggregate $c_l$ and a new header $hdr_l = hdr_i \cup hdr_j$. Suppose $c_i$ and $c_j$ are the ciphertexts for plaintext aggregates $m_i$ and $m_j$ respectively. The output $c_l$ is the ciphertext for the aggregate $f(m_i, m_j)$, namely, $\mathsf{D}_{dk}(hdr_l, c_l) \rightarrow f(m_i, m_j)$. *Note that the aggregation algorithm does not need the decryption key $dk$ or any of the encryption keys $ek_i$ as input; it is a public algorithm.*

Depending on constructions, the aggregation function $f$ could be any associative function, for instance, $f$ could be the sum, multiplicative product, max, etc.. Leveraging on the associativity property, we abuse the notation in this paper: we denote the composition of multiple copies of $f$ simply by $f(m_1, m_2, \ldots, m_i)$ irrespective of the order of aggregation and call it the $f$-aggregate on $m_1, m_2, \ldots, m_i$; to be precise, it should be written as $f(f(f(m_1, m_2), \ldots), m_i)$ with a certain aggregation order. Many aggregation functions of practical interest, such as sum and multiplicative product, satisfy the reversibility property defined as follows.

*Definition 1:* An aggregation function $f$ is reversible if given $x$ and $f(x, y)$, the other input $y$ can be uniquely and efficiently determined.

It is intentional to include the description of the header $hdr$ in the above definition so as to make the CDA security model as general as possible. Nonetheless, generating headers or including headers as input to algorithms should not be treated as a requirement in the actual construction of CDA algorithms. For constructions which do not need headers, all $hdr$'s can simply be treated as the empty set $\phi$ in the security model and the discussions in this paper still apply.

We do not pose restrictions on whether global or local random coins should be used for encryption. If each source generates its random coins individually, the random coins are said to be local; if the random coins are chosen by the sink and broadcast to all source nodes, they are global. Global random coins are usually public. When global random coins are used, we do not pose restriction on the reuse of randomness despite that, in practice, each global random coin is treated as nonce, that is, used once only.

### B. AMAC Syntax

Before describing the syntax of AMAC, we briefly describe that for a normal two-party MAC. A typical MAC scheme is a two-tuple $(MAC, VER)$ where $MAC$ takes the secret key $k$ (shared between the communicating parties) and message $m$ as

input and returns a tag $t = MAC_k(m)$. $VER_k(m,t)$, taking the key $k$, the message $m$ and the tag $t$ as input, returns either 1 (if $t$ is a valid tag for $m$) or 0 (otherwise). For correctness, $VER_k(m, MAC_k(m)) = 1$.

The setting for AMAC is the same as that for CDA, with one sink $R$ and a set $U$ of $n$ source nodes. As before, let $U = \{s_i : 1 \leq i \leq n\}$ and the set of source identities $ID = [1, n]$. Same as in CDA, $hdr \subseteq ID$ is a header indicating the source nodes contributing to an aggregate. Note that in AMAC, the aggregation is done in the plaintext domain (compared to the encrypted domain aggregation in CDA) as the aggregate integrity goal is isolated from the privacy goal in the consideration of AMAC. Without loss of generality, the aggregation function $f(\cdot)$ is assumed to be associative. For a security parameter $\lambda$, an AMAC consists of three polynomial time algorithms as follows.

Key Generation ($\mathsf{KG}$). Let $\mathsf{KG}(1^\lambda, n) \rightarrow (k_1, k_2, \ldots, k_n)$ be a probabilistic algorithm. Then, $k_i$ (with $1 \leq i \leq n$) is the secret key used to generate a verification tag by node $i$. The sink possesses all $k_i$'s used for tag verification.

Tag Generation ($\mathsf{MAC}$). $\mathsf{MAC}_{k_i}(m_i) \rightarrow tag_i$ takes a secret key $k_i$ and a message $m_i$ as input to generate a verification tag $tag_i$ for $m_i$. The message sent out from node $i$ is a 3-tuple $(\{i\}, m_i, tag_i)$.

Tag Verification ($\mathsf{Ver}$). Let $m$ be an $f$-aggregate of messages $m_1, m_2, ..., m_i, ...$ and $hdr$ be the set of all contributing identities. Then $\mathsf{Ver}_{k_1,k_2,...,k_i,...}(m, tag_1, tag_2, ..., tag_i, ...) \rightarrow 0/1$ takes the aggregate $m$ and the tag $tag_i$ and secret key $k_i$ for each $i \in hdr$ and outputs 1 if $m$ is a correct aggregate (i.e. $m = f(m_1, m_2, ..., m_i, ...)$) and 0 otherwise.

Note that no aggregation algorithm is specified in AMAC; the aggregation is done in plaintext, just the same as in usual aggregation. When an aggregating node with identity $k$ receives two measurement values and their tags from downstream, say, $(\{i\}, m_i, tag_i)$ and $(\{j\}, m_j, tag_j.)$, it would pass $(\{i, j, k\}, f(m_i, m_j, m_k), tag_i, tag_j, tag_k)$ as the aggregation result to its parent where $m_k$ is its own measurement. Aggregation of verification tags is not considered here. So all the tags are needed in the verification. Let $m = f(m_1, ..., m_i, ...)$, then the correctness requirement of AMAC is as follows:

$$\mathsf{Ver}_{k_1,...,k_i,...}(m, \mathsf{MAC}_{k_1}(m_1), ..., \mathsf{MAC}_{k_i}(m_i), ...) = 1.$$

**Typical AMAC Operation.** In the initialization phase, the sink generates $n$ secret keys $k_i$'s and gives $k_i$ to node $i$. To respond to a query, each reporting node $s_i \in S \subseteq U$ sends in a data-tag pair $(m_i, tag_i)$ where $tag_i = \mathsf{MAC}_{k_i}(m_i)$. That is, $tag_i$ is supposed to be a verification tag for message $m_i$ generated by node $i$. As these pairs go upstream along the concast tree, aggregation is performed on $m_i$ leaving $tag_i$ intact. Hence, eventually the sink would receive $m$ and all the tags in $\{tag_i = \mathsf{MAC}_{k_i}(m_i) : s_i \in S\}$ where $m$ is supposed to be the $f$-aggregate of all the messages in $\{m_i : s_i \in S\}$. The sink runs the verification algorithm $\mathsf{Ver}$ to check the validity of $m$.

## III. Privacy Notions of CDA

Two types of oracle queries (adversary interaction with the system) are allowed in the security model, namely, the encryption oracle $\mathcal{O}_E$ and the decryption oracle $\mathcal{O}_D$.

Encryption Oracle $\mathcal{O}_E(i, m)$. For fixed encryption and decryption keys, on input an encryption query $\langle i, m\rangle$, the encryption oracle retrieves $s_i$'s encryption key $ek_i$ and runs the encryption algorithm on $m$ and replies with the ciphertext $\mathsf{E}_{ek_i}(m; r)$ and its header $hdr$. In case global random coins are used, the random coins $r$ are part of the query input to $\mathcal{O}_E$.

Decryption Oracle $\mathcal{O}_D(hdr, c)$. For fixed encryption and decryption keys, on input a decryption query $\langle hdr, c\rangle$ (where $hdr \subseteq ID$), the decryption oracle retrieves the decryption key $dk$ and runs the decryption algorithm $\mathsf{D}$ and replies with the result $\mathsf{D}_{dk}(hdr, c)$.

To define security (more precisely, indistinguishability) against adaptive chosen ciphertext attacks ($\mathsf{IND\text{-}CCA2}$), we use the following game between a challenger and an adversary, assuming there is a set $U$ of $n$ source nodes. If no PPT adversary, even in collusion with at most $t$ compromised node (with $t < n$), can win the game with non-negligible advantage (as defined below), we say the CDA scheme is $t$-secure.[1]

*Definition 2:* A CDA scheme is $t$-secure (indistinguishable) against adaptive chosen ciphertext attacks if the advantage of winning the following game is negligible in the security parameter $\lambda$ for all $PPT$ adversaries.

Collusion Choice. The adversary chooses to corrupt $t$ source nodes. Denote the set of these $t$ corrupted nodes and the set of their identities by $S'$ and $I'$ respectively.

Setup. The challenger runs $\mathsf{KG}$ to generate a decryption key $dk$ and $n$ encryption keys $\{ek_i : 1 \leq i \leq n\}$, and gives the subset of $t$ encryption keys $\{ek_j : s_j \in S'\}$ to the adversary but keeps the decryption key $dk$ and the other $(n-t)$ encryption keys $\{ek_j : s_j \in U \backslash S'\}$.

Query 1. The adversary can issue to the challenger two types of queries: [2]

- Encryption Query $\langle i_j, m_j\rangle$. The challenger responds with $\mathsf{E}_{e_{i_j}}(m_j)$.
- Decryption Query $\langle hdr_j, c_j\rangle$. The challenger responds with $\mathsf{D}_{dk}(hdr_j, c_j)$.

Challenge. Once the adversary decides that the first query phase is over, it selects a subset $S$ of $d$ source nodes (whose identities are in the set $I$) such that $|S \cap S'| = 0$, and outputs two different sets of plaintexts $M_0 = \{m_{0k} : k \in I\}$ and $M_1 = \{m_{1k} : k \in I\}$ to be challenged. The only constraint is

[1] The adversary is allowed to freely choose parameters $n$ and $t$.

[2] In case global random coins are used, the adversary is allowed to choose and submit his choices of random coins for both encryption and decryption queries. Depending on whether the encryption keys are kept secret, the encryption queries may or may not be needed.

that the two resulting plaintext aggregates $x_0$ and $x_1$ are not equal where $x_0 = f(\ldots, m_{0k}, \ldots)$ and $x_1 = f(\ldots, m_{1k}, \ldots)$.

The challenger flips a coin $b \in \{0,1\}$ to select between $x_0$ and $x_1$. The challenger then encrypts each $m_{bk} \in M_b$ with $ek_k$ and aggregates the resulting ciphertexts in the set $\{\mathsf{E}_{ek_k}(m_{bk}) : k \in I\}$ to form the ciphertext $C$ of the aggregate, that is, $\mathsf{D}_{dk}(I, C) = x_b$, and gives $(I, C)$ as a challenge to the adversary. In case global random coins are used for encryption, the challenger chooses and passes them to the adversary. If a *nonce* is used, the global random coins should be chosen different from those used in the Query 1 phase and no encryption query on them should be allowed in the Query 2 phase.

Query 2. The adversary is allowed to make more queries (both encryption and decryption) as previously done in Query 1 phase but for decryption queries, $hdr_j \subseteq S$ and no decryption query can be made on the challenged ciphertext $C$ if $hdr_j = S$.

Guess. Finally, the adversary outputs a guess $b' \in \{0,1\}$ for $b$.

Result. The adversary wins the game if $b' = b$. The advantage of the adversary is defined as: $Adv_{\mathcal{A}} = \left|Pr[b' = b] - \frac{1}{2}\right|$.

Note that in CDA what the adversary is interested in is the information about the final aggregate. Consequently, in the above game, the adversary is asked to distinguish between the ciphertexts of two *different* aggregates $x_0$ and $x_1$ as the challenge, rather than to distinguish two different sets of plaintexts $M_0$ and $M_1$. By picking elements for $M_0$ and $M_1$, the adversary is essentially free to choose $x_0$ and $x_1$. Allowing the adversary to choose the two sets $M_0, M_1$ is to give him more flexibility in launching attacks.

The above definition of security against CCA2 attacks is a weaker version of the original notion defined in [5] in the sense that, in the Query 2 phase, the adversary is more restricted in making decryption queries now. Originally, the adversary can submit a decryption query with a header such that $S$ is a subset of it; this type of query is not allowed now. The reason for such a modification is that the original security notion could not be achieved normally when compromised nodes exist[3] due to the following attack: an adversary can choose a choice of $S$ such that it can add in the contribution from a compromised node not in $S$ to ask for a decryption query in Query 2 phase; through this query result, the adversary can determine which of the two aggregates he is being challenged with; this is possible as long as, $f(x_0, a) \neq f(x_1, a)$ where $a$ is the newly added contribution. This attack is achieved through the manipulation of the aggregation functionality. Due to the same attack, a straightforward extension of MAC for AMAC cannot be achieved (Section IV).

[3]The CMT scheme and its variant [4], [5] can achieve this notion of indistinguishability against CCA2 attacks when employing stateful decryption to ensure the same reply for each nonce in decryption queries.

It should be noted that even for this weakened notion of CCA2 security, no scheme has been constructed to achieve it so far. Existing constructions in the literature [12], [4], [5] are insecure in this security model. It is fair to say constructing schemes to achieve this privacy notion remains open.

## IV. Security Notions of AMAC

Two types of oracle queries are allowed in the AMAC security model, namely, the tag generation oracle $\mathcal{O}_T$ and the tag verification oracle $\mathcal{O}_V$. Their details are as follows:

Tag Generation Oracle $\mathcal{O}_T(i, m)$. For fixed secret keys, on input a tag generation query $\langle i, m\rangle$, the tag generation oracle retrieves key $k_i$ to run the tag generation algorithm on $m$ and replies with the tag $\mathsf{MAC}_{k_i}(m)$.

Tag Verification Oracle $\mathcal{O}_V(hdr, m, T = \{tag_i : i \in hdr\})$. For fixed secret keys, on input a tag verification query $\langle hdr, m, T\rangle$ (where $hdr \subseteq ID$ and $T = \{tag_i : i \in hdr\}$), the tag verification oracle retrieves the keys $\{k_i : i \in hdr\}$ and runs the tag verification algorithm $\mathsf{Ver}$ and replies with the result $\mathsf{Ver}_{\{k_i : i \in hdr\}}(m, T)$.

We adopt a notion of existential unforgeability against chosen message attacks for AMAC security.

*Definition 3:* An AMAC scheme is $t$-secure (unforgeability) against chosen message attacks if the advantage of winning the following game is negligible in the security parameter $\lambda$ for all $PPT$ adversaries.

Collusion Choice.The adversary chooses to corrupt $t$ source nodes. Denote the set of these $t$ corrupted nodes and the set of their identities by $S'$ and $I'$ respectively.

Setup. The challenger runs $\mathsf{KG}$ to generate $n$ secret keys $\{k_i : 1 \leq i \leq n\}$, and gives the subset of $t$ keys $\{k_j : s_j \in S'\}$ to the adversary but keeps the other $n - t$ keys $\{k_j : s_j \in U \backslash S'\}$.

Query. The adversary can issue two types of queries:

- Tag Generation Query $\langle i_j, m_j\rangle$. The challenger responds by returning the tag $\mathsf{MAC}_{k_{i_j}}(m_j)$.
- Tag Verification Query $\langle hdr_j, m_j, T_j\rangle$. The challenger responds by returning the verification result $\mathsf{Ver}_{\{k_l : l \in hdr_j\}}(m_j, T_j)$.

The adversary can issue queries of both types until he decides to make a guess. Denote the set of tag generation queries made so far by $\mathcal{T}_Q = \{\langle i_j, m_j\rangle\}$. Let $\mathcal{M}_Q = \{f(m_{j_1}, ..., m_{j_x}, m_{j_y}, ...,) : \langle i_{j_x}, m_{j_x}\rangle \in \mathcal{T}_Q, \langle i_{j_y}, m_{j_y}\rangle \in \mathcal{T}_Q; i_{j_x} \neq i_{j_y}, \forall x, \forall y)\}$. That is, $\mathcal{M}_Q$ denote the set of all possible aggregates which can be obtained from aggregating messages in the tag generation queries.

Guess. Finally, the adversary outputs an aggregate and the associated verification tags, that is, a tuple $\langle hdr, m, T = \{tag_i : i \in hdr\}\rangle$ where $hdr \subseteq U \backslash S'$ and $m \notin \mathcal{M}_Q$.

Result. The adversary wins the game if $\mathsf{Ver}_{\{k_l : l \in hdr\}}(m, T) = 1$. The advantage of the adversary is defined as the probability that the adversary wins the game.

A MAC scheme is normally considered as a symmetric key counterpart of digital signatures. The security requirement for MAC schemes is in essence the same as that for digital signatures, namely, unforgeability against chosen message attacks [2], [8]. In details, the secret key $k$ is kept secret from an adversary. The adversary is allowed to query tags $t_j$ for messages $m_j$ of his choice, which can be done adaptively. Let $\mathcal{M}$ denote the set of $m_j$'s made in the tag generation query. The adversary breaks the scheme if he is able to find a message $m \notin \mathcal{M}$ and a valid tag $t = MAC_k(m)$.

In the Guess phase in the simulation game above, we require that $|hdr \cap S'| = 0$ (as can be implied from $hdr \subseteq U \backslash S'$). Compared with the security notion for MAC, it is natural to ask whether this requirement is necessary as the requirement $m \notin \mathcal{M}_Q$ seems enough and is a natural refinement of the security model for MAC. In particular, this additional requirement leads to a weaker security model, namely, a more difficult task for the adversary is assumed. Without imposing the requirement that $hdr \subseteq U \backslash S'$, it can be shown that when compromised nodes exist, all constructions would be insecure[4] in the AMAC security model due to the aggregation functionality, in particular when no stateful tag generation is assumed. In practice, this requirement means that an adversary cannot deviate the final aggregate beyond what can be achieved by injecting contributions directly into the aggregate through compromised nodes. In other words, an adversary can only aggregate messages and inject its contributions as what a honest node does, and he cannot modify the contributions (in the aggregate) from honest nodes downstream in the concast tree without being detected by the sink. Nevertheless, whether this security notion for AMAC can be achieved remains unknown. The knowledge about this AMAC security notion is summarized by the following claim.

**Claim.** *Assuming the existence of one-way functions, if an AMAC construction secure in the sense of Definition 3 exists, an IND-CCA2-secure CDA construction as defined in Definition 2 exists for any reversible aggregation function $f$ (as defined in Definition 1).*

**Proof Sketch** (Informal). Assume there exists a secure construction of AMAC. We can use this AMAC primitive and any semantic-secure CDA scheme $\mathcal{E}$ to construct an IND-CCA2-secure CDA scheme. The construction is intuitive: to encrypt a message $m$, we generate a random coin $r$ and use $\mathcal{E}$ to encrypt $r$ and $f(r, m)$; then two AMAC tags, on $r$ and $f(r, m)$ respectively, are attached to the two ciphertexts. While the ciphertexts are aggregated, no aggregation on the AMAC tags is done. In decryption, we decrypt the two ciphertext aggregates and verify that the aggregates and the corresponding AMAC tags match using Ver. If the result is 1, we output the decrypted message, otherwise, we output $\perp$ (a decryption failure). The resulting scheme could resist CCA2 attacks in Definition 2 mainly because, if an adversary modifies the challenged ciphertext to make a decryption query, with high probability it will be caught and gain no useful information from $\perp$. Since $f$ is reversible, we can determine the message aggregate from the two final aggregates, $f(..., r_i, ...)$ and $f(..., r_i + m_i, ...)$. The technique in [4], [5] can be used to construct a semantic-secure CDA for $f$ from a pseudorandom function (which in turn can be constructed from a one-way function). ■

Since no CDA construction in the literature achieves IND-CCA2 security, it may be that such CDA constructions do not exist. If it is the case, then the AMAC security defined in Definition 3 cannot be achieved.

[4]This AMAC security notion cannot be achieved by any constructions. If such an AMAC scheme exists, it can help to construct a CDA scheme achieving the IND-CCA2 security in strong sense as defined in [5] which may not be achievable.

## V. Conclusions

End-to-end privacy and integrity/authenticity are the two main goals of secure data aggregation. We give security models for both privacy and aggregate integrity and derive relations between them.

## Acknowledgment

The first author would like to acknowledge the financial support provided by the Ministry of Education, Singapore through the Lee Kuan Yew Postdoctoral Fellowship and AcRF research grant R-252-000-331-112.

## References

[1] M. Acharya, J. Girao, and D. Westhoff. Secure comparison of encrypted data in wireless sensor networks. In *the Proceedings of WiOpt 2005*, April 2005.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology — CRYPTO 1996, Springer-Verlag LNCS vol. 1109*, pages 1–15, 1996.

[3] D. Boneh, D. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear map. In *Advances in Cryptology — EUROCRYPT 2003, Springer-Verlag LNCS vol. 2656*, pages 272–293, 2003.

[4] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *the Proceedings of MobiQuitous'05*, pages 1–9, July 2005.

[5] Aldar C-F. Chan and Claude Castelluccia. On the privacy of concealed data aggregation. In *ESORICS 2007, Springer-Verlag LNCS vol. 4734*, September 2007.

[6] Haowen Chan, Adrian Perrig, and Dawn Song. Secure hierarchical in-network aggregation in sensor networks. In *ACM Conference on Computer and Communication Security (CCS 06*, October 2006.

[7] J. Girao, D. Westhoff, and M. Schneider. CDA: Concealed data aggregation in wireless sensor networks. In *the Proceedings of IEEE International Conference on Communication(ICC'05)*, May 2005.

[8] S. Goldwasser, S. Micali, and R. Rivest. A secure signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[9] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, 2003.

[10] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. In *Fast Software Encryption (FSE 2003), Springer-Verlag LNCS vol. 2887*, pages 129–153, 2003.

[11] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *the Proceedings of 1st International Conference on Embedded Networked Sensor Systems*, 2003.

[12] D. Westhoff, J. Girao, and M. Acharya. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaption. *IEEE Transactions on Mobile Computing*, 5(10):1417–1431, 2006.

[13] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *the Proceedings of ACM Internation Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc) 2006*, 2006.